Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЁТ

По лабораторной работе №7

По курсу: «Функциональное и логическое программирование»

Студент: Керимов А. Ш.

Группа: ИУ7-64Б

Преподаватели: Толпинская Н. Б.,
Строганов Ю. В.

Москва

2020

Задание 1. Пусть `(setf lst1 '(a b)) (setf lst2 '(c d))`. Каковы результаты вычисления следующих выражений?

1. `(cons lst1 lst2) ; ((A B) C D)`
2. `(list lst1 lst2) ; ((A B) (C D))`
3. `(append lst1 lst2) ; (A B C D)`

Задание 2. Каковы результаты вычисления следующих выражений?

1. `(reverse ()) ; NIL`
2. `(last ()) ; NIL`
3. `(reverse '(a)) ; (A)`
4. `(last '(a)) ; (A)`
5. `(reverse '((a b c))) ; ((A B C))`
6. `(last '((a b c))) ; ((A B C))`

Задание 3. Написать, по крайней мере, два варианта функции, которая возвращает последний элемент своего списка-аргумента.

```
(defun my-last1 (x)
  (car (reverse x)))

(defun my-last2 (x)
  (cond ((null (cdr x)) (car x))
        (T (my_last2 (cdr x)))))

(defun my-last3 (x)
  (reduce (lambda (p n) n)))
```

Задание 4. Написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента.

```
(defun lastout1 (x)
  (reverse (cdr (reverse x))))

(defun lastout2 (x)
  (butlast x))
```

Задание 5. Написать простой вариант игры в кости, в котором бросаются две правильные кости. Если сумма выпавших очков равна 7 или 11 — выигрыш, если выпало (1, 1) или (6, 6) — игрок получает право снова бросить кости, во всех остальных случаях ход переходит ко второму игроку, но запоминается сумма выпавших очков. Если второй игрок не выигрывает абсолютно, то выигрывает тот игрок, у которого больше очков.

```
(defun throw-one () (+ (random 6) 1))

(defun throw-two () (list (throw-one) (throw-one)))

(defun next-try (dices)
  (or (equal dices '(1 1))
      (equal dices '(6 6))))

(defun sum-dices (dices)
  (+ (car dices) (cadr dices)))

(defun is-win (dices)
  (let ((rate (sum-dices dices)))
```

```
       (or (= rate 7) (= rate 11))))

(defun play-round ()
  (let ((dices (throw-two)))
       (cond
         ((is-win dices) (list 0 dices))
         ((next-try dices) (play-round))
         (T (list (sum-dices dices) dices)))))

(defun print-sum (d1 d2 msg)
  (format nil "Player 1 - ~{~a~^ ~}~%Player 2 - ~{~a~^ ~}~%~a"
    (cdr d1) (cdr d2) msg))

(defun play ()
  (let ((d1 (play-round))
        (d2 (play-round)))
       (cond
         ((= (car d1) 0) (print-sum d1 d2 "Player 1 won!"))
         ((= (car d1) 0) (print-sum d1 d2 "Player 2 won!"))
         ((> (car d1) (car d2)) (print-sum d1 d2 "Player 1 won (sum)!"))
         ((< (car d1) (car d2)) (print-sum d1 d2 "Player 2 won (sum)!"))
         (T (print-sum d1 d2 "Draw!")))))
```